[0100] By treating the communication medium as an object, the system **20** offers strategies to support data transfer that are in compliance with the policy. For example, the producer process could create an object that represents the physical medium/support of the data transfer mechanism. This new object may be assigned attributes in accordance to a predefined and policy-specific set of conditions. These conditions can be specified via the event-response relations (e.g., if a process reads secret data, any subsequently created object will be assigned to the secret attribute). The consumer process must be able to read the object that represents the physical medium under the rules of the reference mediation.

[0101] A practical example pertains to the clipboard that is used in performing copy/cut and paste operations. When process p_1 issues the copy/cut request, the access control module **28** creates an object, say co, which represents the clipboard, with attributes that are specified in the event-response relations. The data is transferred to the clipboard as usual. When the same or different process p_2 issues the paste request, the access control module **28** naturally treats this request as a request to read from object co. As for any other process request, the access control module **28** invokes the reference mediation function to check whether p_2 is authorized to read the object co. If the request is granted by the reference mediation function, p_2 continues with the paste operation as usual.

[0102] The following examples illustrate the ability of the system **20** to support the security objectives of RBAC and MLS security models. The system **20** does not necessarily emulate the specific rules or relations of any model, but rather is able to achieve the same policy objectives of those models.

[0103] Consider the combination of the two access control policies depicted in FIG. **3**, a role-based access control (RBAC) policy, and a multi-level security (MLS) policy. A known administrative objective of RBAC is to streamline authorization management by defining roles as relations between users and capabilities. These relations are achieved by assigning users to roles on one side and assigning capabilities to roles on the other side. By assigning a user to a role, that user instantaneously acquires the capabilities that are assigned to the role. Another RBAC feature is the ability to define a role hierarchy, i.e., an inheritance relation between roles, whereby senior roles acquire the capabilities of their juniors. By assigning a user to a role, the user is also (indirectly) associated with the capabilities of that role's junior roles. Finally, the RBAC standard includes two types of relations for the enforcement of separation of duties: static separation of duty (SSoD) and dynamic separation of duty (DSoD).

[0104] The system **20** meets the administrative and policy objectives of RBAC. Indeed, a user attribute in the system **20** includes the semantics of a RBAC role, i.e. by assigning a user to that user attribute, the user acquires the capabilities associated with the user attribute. Moreover, the system **20** is superior to RBAC in administrative efficiency, due to additional abstractions. In the system **20**, capabilities are indirectly associated with user attributes through the double assignment user attribute-operation set-object attribute. By assigning a user to a user attribute the user is capable of performing the operations in the operation set on the objects in the container represented by the object attribute. Furthermore, the system **20** allows for the efficient association of objects with access control entries of the form (user, operation), while RBAC offers no semantics in this regard. With regard to role hierarchies, the system **20** offers semantics similar to RBAC through the user attributes assignments to other user attributes. Finally, the system **20** allows for the inheritance of access control entries between object attributes.

[0105] Considering the example in FIG. **3**, the user attributes Doctor, Intern, and Consultant represent RBAC roles. The configuration includes object attributes Med Records and Development, and objects like mrec**1** and project**1**. Under the RBAC policy, user alice's permissions are directly derived from alice's assignment to the user attribute Doctor (i.e., (alice, w, mrec**1**), (alice, w, mrec**2**), and (alice, w, mrec**3**)). Alice also inherits the permissions (alice, r, mrec**1**), (alice, r, mrec**2**), (alice, r, mrec**3**) from the assignment Doctor→Intern, which offers the same semantics as that of the role hierarchy.

[0106] Conflict of interest in a role-based system may arise as a result of a user gaining authorization for capabilities associated with conflicting roles. One means of preventing this form of conflict of interest is through static separation of duty (SSOD) to enforce constraints on the assignment of users to roles. SSoD relations place constraints on the assignments of users to roles. Membership in one role may prevent the user from being a member of one or more other roles. Dynamic separation of duty (DSoD) relations, like SSoD relations, limits the capabilities that are available to a user. However DSoD relations differ from SSD relations by the context in which these limitations are imposed. DSoD requirements limit the availability of the capabilities by placing constraints on the roles that can be activated within or across a user's sessions.

[0107] The system **20** can be programmed to provide the same security objectives as SSoD and DSoD, but through different means. Assume that a conflict of interest would arise if a user were able to execute capability (op_1, o_1) and capability (op_2, o_2). Under RBAC, these capabilities would be assigned to different roles (say r_1 and r_2) and an SSoD relation would be imposed between those roles and thus prevent any user from being simultaneously assigned to both roles. The following obligation relations in system **20** can be used to achieve this same objective:

[0108] process performs (op_1, o_1) ⇒deny (process user, {op_2}, o_2)

[0109] process performs (op_2, o_2) ⇒deny (process user, {op_1}, o_1)

[0110] Through these relations, any process that successfully executes (op_1, o_1) would effectively deny the process user the ability to successfully execute (op_2, o_2) in the future and vice-versa. Furthermore, in an RBAC SSD environment, while a user u_1 that is assigned to r_1 would be prevented from executing (op_2, o_2) through denial of membership to r_2, nothing prevents (op_2, o_2) from being assigned to some r_3 and u_1 being assigned to r_3.

[0111] Regarding DSoD, assume once again the capability (op_1, o_1) and capability (op_2, o_2) that are respectively assigned to r_1 and r_2. Also, assume r_1 and r_2 are in a DSoD relation in RBAC. Under these circumstances, no user may have the ability to execute both capabilities within the same session. However, a user can assume both roles in different sessions either concurrently or sequentially. Given that DSoD does not limit a user's ability to execute both capabilities within different session, we see the security objective as being that of enforcement of least privilege at the process level. Under least privilege a process should be prevented from executing (either maliciously or by error) both capabilities. The following obligation relation in system **20** can be used to achieve this same objective:

[0112] process performs (op_1, o_1)⇒deny (current process, {op_2}, o_2)

[0113] process performs (op_2, o_2)⇒deny (current process, {op_1}, o_1).